ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΠΡΟΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ
ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ

ΑΛΓΟΡΙΘΜΟΙ

ΔΙΑΛΕΞΗ 7

ΔΙΔΑΣΚΩΝ
ΚΩΣΤΑΣ ΚΟΛΟΜΒΑΤΣΟΣ

Ασκήσεις στους Αλγορίθμους Ταξινόμησης

Άσκηση 1

Υποθέτουμε ότι το n είναι πολλαπλάσιο του 2 και ότι ένας αλγόριθμος ταξινόμησης χαρακτηρίζεται από την αναδρομική σχέση

$$T(n) = \begin{cases} 2 & \text{if } n = 2\,, \\ 2T(n/2) + n & \text{if } n = 2^k, \text{ for } k > 1 \end{cases}$$

Να αποδείξετε ότι

$$T(n) = n \lg n$$

Λύση

$$T(n) = \begin{cases} 2 & \text{if } n = 2\,, \\ 2T(n/2) + n & \text{if } n = 2^k, \text{for } k > 1 \end{cases}$$

Η καλύτερη περίπτωση συναντάται όταν n=2 οπότε έχουμε:

nlogn = 2log2=2

Για την αναδρομική εξίσωση έχουμε ότι T(n/2)=(n/2)log(n/2)

Οπότε

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2(n/2)\lg(n/2) + n \\ &= n(\lg n - 1) + n \\ &= n\lg n - n + n \\ &= n\lg n \end{aligned}$$

Άσκηση 2

Ο αλγόριθμος insertion sort μπορεί να υλοποιηθεί αναδρομικά ως εξής: Για να ταξινομηθεί ο πίνακας A[1..n], ταξινομούμε αναδρομικά τον υπο-πίνακα A[1..n-1] και στη συνέχεια εισάγουμε το στοιχείο A[n] στη θέση του στον ταξινομημένο υπο-πίνακα A[1..n-1].

Να γράψετε την αναδρομική σχέση για αυτό τον αλγόριθμο.

Λύση

Στη χειρότερη περίπτωση θα χρειαστούμε χρόνο Θ(n) για να τοποθετήσουμε το A[n] στη θέση του. Συνεπώς, η αναδρομική σχέση θα είναι:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(n-1) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Μέσω της οποίας μπορούμε εύκολα να αποδείξουμε πως T(n) = $\Theta(n^2)$

T(n-1)+n-1 […… n-1 ……..] ← A[n]

T(n-2)+n-2+n-1 […… n-2 ……..] ← A[n-1] ← A[n]

Άσκηση 3

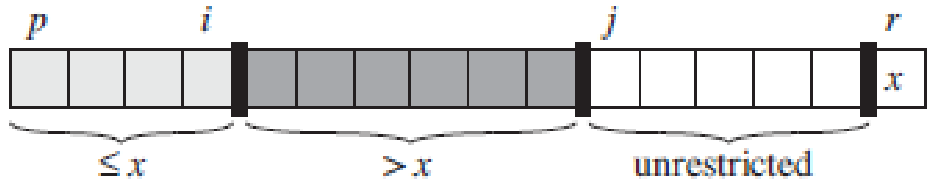
Να αποδείξετε ότι η πολυπλοκότητα του quicksort είναι $\Theta(n^2)$ όταν ο πίνακας Α περιέχει διακριτά στοιχεία (δεν υπάρχουν δύο ίσα στοιχεία μεταξύ τους) και είναι ταξινομημένα σε φθίνουσα σειρά.

Λύση

Αλγόριθμος quick sort

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```



Λύση

Στην περίπτωση που τα στοιχεία είναι σε φθίνουσα σειρά, ο αλγόριθμος PARTITION θα εκτελεί λειτουργία χειρότερης περίπτωσης.

Θα μειώνει το μέγεθος του υπο-πίνακα μόνο κατά 1 σε κάθε βήμα

Για ένα υπο-πίνακα A[p..r], θα παράξει ένα κενό τμήμα στο A[p..r-1] με μόνο ένα λιγότερο στοιχείο σε σχέση με τον αρχικό υποπίνακα

Θα τοποθετεί το pivot στην θέση A[r] αρχικά και μετά στην θέση A[p] και θα παράξει το τμήμα A[p+1..r]

Συνεπώς, η αναδρομική σχέση θα είναι $T(n)=T(n-1) + \Theta(n)$ που έχει πολυπλοκότητα **????**

Άσκηση 4

Να αποδείξετε ότι η πολυπλοκότητα του quick sort στην καλύτερη περίπτωση είναι Ω(nlogn)

Λύση

Έχουμε την αναδρομική σχέση:

$$T(n) = \min_{1 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n)$$

Υποθέτουμε ότι

$$T(n) \ge cn \lg n$$

Για κάποια σταθερά c

Με αντικατάσταση στην αναδρομική σχέση παίρνουμε

$$\begin{aligned} T(n) &\ge \min_{1 \le q \le n-1} (cq \lg q + c(n-q-1) \lg(n-q-1)) + \Theta(n) \\ &= c \cdot \min_{1 \le q \le n-1} (q \lg q + (n-q-1) \lg(n-q-1)) + \Theta(n) \end{aligned}$$

Λύση

Θα πρέπει να δείξουμε πως η σχέση

$$q \lg q + (n - q - 1) \lg(n - q - 1)$$

Έχει ρίζα – ελάχιστο στο

$$1 \leq q \leq n-1$$

Όταν

$$q = n-q-1$$

ή

$$q = (n-1)/2$$

Λύση

Η πρώτη παράγωγος της συνάρτησης είναι 0 όταν

$$q = (n-1)/2$$

Και η δεύτερη παράγωγος είναι θετική

Αν λοιπόν επιλέξουμε $q = (n-1)/2$

τότε $\min_{1 \le q \le n-1} (q \lg q + (n-q-1) \lg(n-q-1)$

$$\geq \frac{n-1}{2} \lg \frac{n-1}{2} + \left(n - \frac{n-1}{2} - 1\right) \lg \left(n - \frac{n-1}{2} - 1\right)$$

$$= (n-1) \lg \frac{n-1}{2}$$

Λύση

Αν τώρα προσπαθήσουμε να φράξουμε το T(n) από κάτω θα έχουμε για n>=2

$$
\begin{aligned}
T(n) &\geq c(n-1)\lg\frac{n-1}{2} + \Theta(n) \\
&= c(n-1)\lg(n-1) - c(n-1) + \Theta(n) \\
&= cn\lg(n-1) - c\lg(n-1) - c(n-1) + \Theta(n) \\
&\geq cn\lg(n/2) - c\lg(n-1) - c(n-1) + \Theta(n) \\
&= cn\lg n - cn - c\lg(n-1) - cn + c + \Theta(n) \\
&= cn\lg n - (2cn + c\lg(n-1) - c) + \Theta(n) \\
&\geq cn\lg n
\end{aligned}
$$

Λύση

Πρέπει να επιλεγεί ένα c αρκετά μικρό ώστε ο όρος Θ(n) να ‘υπερισχύει’ του όρου

$$2cn + c\lg(n-1) - c$$

Οπότε η πολυπλοκότητα της καλύτερης περίπτωσης είναι Ω(nlogn)

Λύση

Όσον αφορά στην παραγώγιση έχουμε:

$$f(q) = q \lg q + (n - q - 1) \lg(n - q - 1)$$

$$f(q) = \frac{q \ln q + (n - q - 1) \ln(n - q - 1)}{\ln 2}$$

$$\begin{aligned} f'(q) &= \frac{d}{dq}\left(\frac{q \ln q + (n - q - 1) \ln(n - q - 1)}{\ln 2}\right) \\ &= \frac{\ln q + 1 - \ln(n - q - 1) - 1}{\ln 2} \\ &= \frac{\ln q - \ln(n - q - 1)}{\ln 2} \end{aligned}$$

Λύση

Η παράγωγος γίνεται 0 όταν

$q = n - q - 1$ ή $q = (n-1)/2$

Και

$$
\begin{aligned}
f''(q) &= \frac{d}{dq}\left(\frac{\ln q - \ln(n-q-1)}{\ln 2}\right) \\
&= \frac{1}{\ln 2}\left(\frac{1}{q} + \frac{1}{n-q-1}\right) \\
f''\left(\frac{n-1}{2}\right) &= \frac{1}{\ln 2}\left(\frac{2}{n-1} + \frac{2}{n-1}\right) \\
&= \frac{1}{\ln 2} \cdot \frac{4}{n-1} \\
&> 0
\end{aligned}
$$

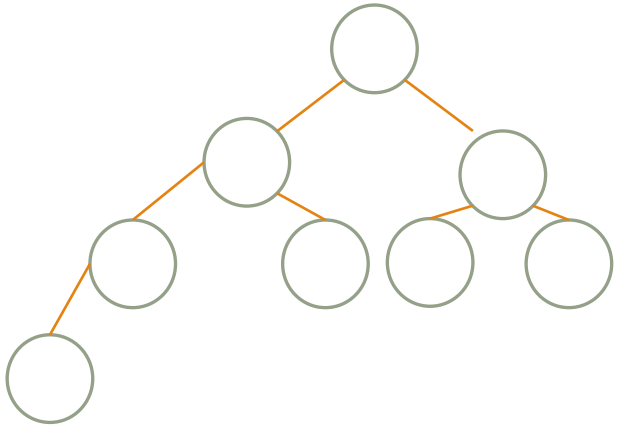
Άσκηση 5

Ποιο είναι το ελάχιστο και το μέγιστο πλήθος των στοιχείων σε ένα σωρό ύψους h?

Λύση

Αφού ο σωρός είναι σχεδόν πλήρες δυαδικό δένδρο, η συμπλήρωση όλων των στοιχείων θα έχει το πολύ $\mathbf{2^h - 1}$ στοιχεία.

Τα στοιχεία θα είναι τουλάχιστον $\mathbf{2^{h-1} - 1 + 1 = 2^{h-1}}$ (όταν το τελευταίο επίπεδο έχει ένα στοιχείο και όλα τα υπόλοιπα επίπεδα είναι γεμάτα)

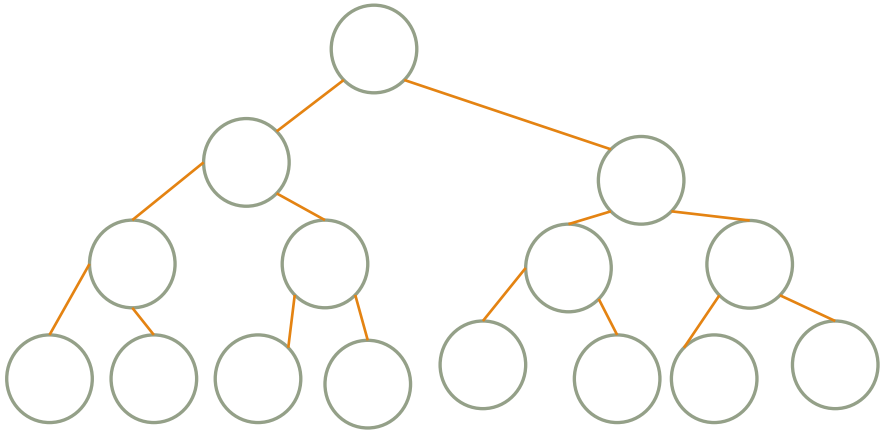


1ο (0)

2ο (1)

3ο (2)

4ο (3)



Άσκηση 5

Να αποδείξετε ότι ένα σωρός που έχει n στοιχεία έχει ύψος **floor(logn)**

Λύση

Με n στοιχεία, έχουμε από την προηγούμενη άσκηση ότι

$2^h <= n <= 2^{h+1}$

Αν λογαριθμήσουμε κατά μέρη θα πάρουμε:

$h <= \log n < h+1$

Οπότε αφού το h είναι ακέραιος έχουμε πως

h = floor(logn)

Άσκηση 6

Να ταξινομήσετε τη λίστα E X A M P L E με τον αλγόριθμο selection sort

Λύση

$$
\begin{array}{|ccccccc}
E & X & \mathbf{A} & M & P & L & E \\
A & X & \mathbf{E} & M & P & L & E \\
A & E & X & M & P & L & \mathbf{E} \\
A & E & E & M & P & \mathbf{L} & X \\
A & E & E & L & P & \mathbf{M} & X \\
A & E & E & L & M & \mathbf{P} & X \\
A & E & E & L & M & P & X
\end{array}
$$

Άσκηση 7

Να ταξινομήσετε τη λίστα E X A M P L E με τον αλγόριθμο bubble sort

Λύση

$$E, X, A, M, P, L, E$$

$$
\begin{array}{ccccccccccccc}
E & \overset{?}{\leftrightarrow} & X & \overset{?}{\leftrightarrow} & A & & M & & P & & L & & E \\
E & & A & & X & \overset{?}{\leftrightarrow} & M & & P & & L & & E \\
E & & A & & M & & X & \overset{?}{\leftrightarrow} & P & & L & & E \\
E & & A & & M & & P & & X & \overset{?}{\leftrightarrow} & L & & E \\
E & & A & & M & & P & & L & & X & \overset{?}{\leftrightarrow} & E \\
E & & A & & M & & P & & L & & E & & |X \\
E & \overset{?}{\leftrightarrow} & A & & M & & P & & L & & E & & \\
A & & E & \overset{?}{\leftrightarrow} & M & \overset{?}{\leftrightarrow} & P & \overset{?}{\leftrightarrow} & L & & E & & \\
A & & E & & M & & L & & P & \overset{?}{\leftrightarrow} & E & & \\
A & & E & & M & & L & & E & & |P & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & M & \overset{?}{\leftrightarrow} & L & & E & & & & \\
A & & E & & L & & M & \overset{?}{\leftrightarrow} & E & & & & \\
A & & E & & L & & E & & |M & & & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & L & \overset{?}{\leftrightarrow} & E & & & & & & \\
A & & E & & E & & |L & & & & & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & L & & & & & &
\end{array}
$$

Άσκηση 8

Ο selection sort είναι ένας σταθερός (stable) αλγόριθμος?

Λύση

Η απάντηση είναι όχι!

Σκεφτείτε την ακολουθία 2, 2, 1

Ο αλγόριθμος, συνήθως, αντιμεταθέτει μη γειτονικά στοιχεία

Άσκηση 9

Ο bubble sort είναι ένας σταθερός (stable) αλγόριθμος?

Λύση

Η απάντηση είναι ναι!

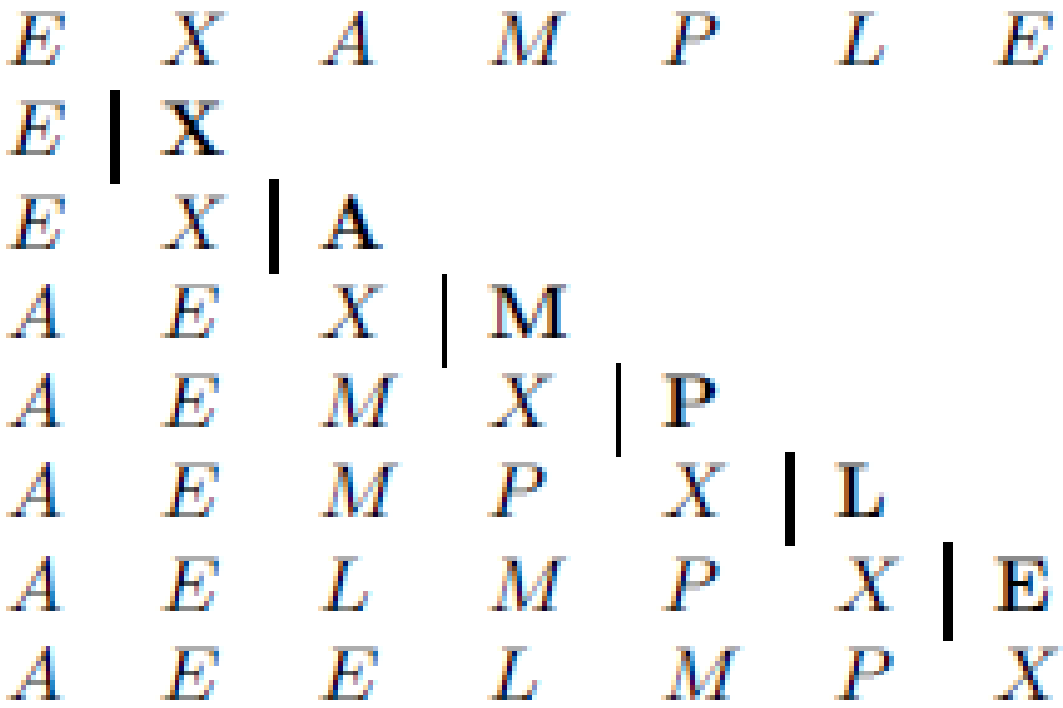
Ο αλγόριθμος πάντα συγκρίνει γειτονικά στοιχεία μέσω της συνθήκης $A[j+1] < A[j]$

Συνεπώς σε ισότητα δεν αντιμεταθέτει τα στοιχεία και παραμένουν στη σειρά τους

Άσκηση 10

Να ταξινομήσετε τη λίστα E X A M P L E με τον αλγόριθμο insertion sort

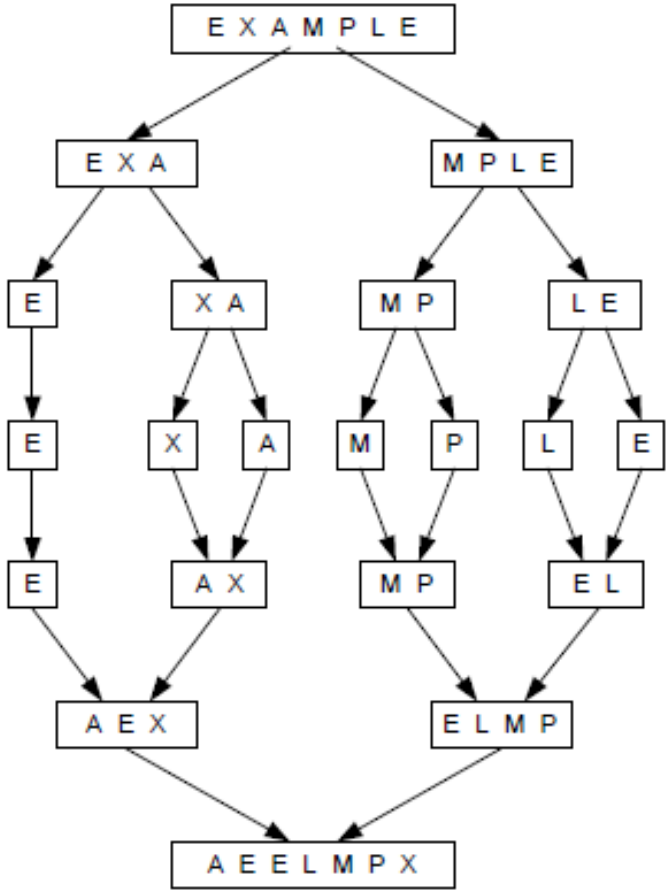
Λύση



Άσκηση 11

Να ταξινομήσετε τη λίστα E X A M P L E με τον αλγόριθμο merge sort

Λύση



Άσκηση 12

Να επιλύσετε την αναδρομική σχέση των συγκρίσεων στον merge sort στη χειρότερη περίπτωση

Λύση

Η σχέση έχει δοθεί στην προηγούμενη διάλεξη και είναι

$$C_{worst}(n) = 2C_{worst}(n/2) + n - 1 \quad \text{for } n > 1, \quad C_{worst}(1) = 0$$

Η λύση της έχει ως εξής (για n άρτιο):

$$\begin{aligned}
C_w(2^k) &= 2C_w(2^{k-1}) + 2^k - 1 \\
&= 2[2C_w(2^{k-2}) + 2^{k-1} - 1] + 2^k - 1 = 2^2 C_w(2^{k-2}) + 2 \cdot 2^k - 2 - 1 \\
&= 2^2[2C_w(2^{k-3}) + 2^{k-2} - 1] + 2 \cdot 2^k - 2 - 1 = 2^3 C_w(2^{k-3}) + 3 \cdot 2^k - 2^2 - 2 - 1 \\
&= \ldots \\
&= 2^i C_w(2^{k-i}) + i2^k - 2^{i-1} - 2^{i-2} - \ldots - 1 \\
&= \ldots \\
&= 2^k C_w(2^{k-k}) + k2^k - 2^{k-1} - 2^{k-2} - \ldots - 1 = k2^k - (2^k - 1) = n \log n - n + 1
\end{aligned}$$

Άσκηση 13

Να ορίσετε μια αναδρομική σχέση για το πλήθος των συγκρίσεων του merge sort στην καλύτερη περίπτωση

Λύση

Στην καλύτερη περίπτωση ο πίνακας θα είναι ήδη ταξινομημένος. Συνεπώς η αναδρομική σχέση είναι:

$$C_b(n) = 2C_b(n/2) + n/2 \quad \text{for } n > 1\ , \quad C_b(1) = 0$$

Η λύση της έχει ως εξής (για n άρτιο):

$$\begin{aligned}
C_b(2^k) &= 2C_b(2^{k-1}) + 2^{k-1} \\
&= 2[2C_b(2^{k-2}) + 2^{k-2}] + 2^{k-1} = 2^2 C_b(2^{k-2}) + 2^{k-1} + 2^{k-1} \\
&= 2^2[2C_b(2^{k-3}) + 2^{k-3}] + 2^{k-1} + 2^{k-1} = 2^3 C_b(2^{k-3}) + 2^{k-1} + 2^{k-1} + 2^{k-1} \\
&= \ldots \\
&= 2^i C_b(2^{k-i}) + i2^{k-1} \\
&= \ldots \\
&= 2^k C_b(2^{k-k}) + k2^{k-1} = k2^{k-1} = \frac{1}{2} n \log n
\end{aligned}$$

Θεώρημα

Οποιοσδήποτε αλγόριθμος ταξινόμησης απαιτεί Ω(nlogn) στη χειρότερη περίπτωση

Απόδειξη

Για την απόδειξη θα υιοθετήσουμε τα δένδρα απόφασης / σύγκρισης (decision trees) για τους αλγορίθμους ταξινόμησης

Ένα δένδρο απόφασης είναι ένα δένδρο που αναπαριστά τις συγκρίσεις που υλοποιεί ένας αλγόριθμος ταξινόμησης

Έλεγχοι, αντιμεταθέσεις, κ.λπ. αγνοούνται σε ένα δένδρο απόφασης

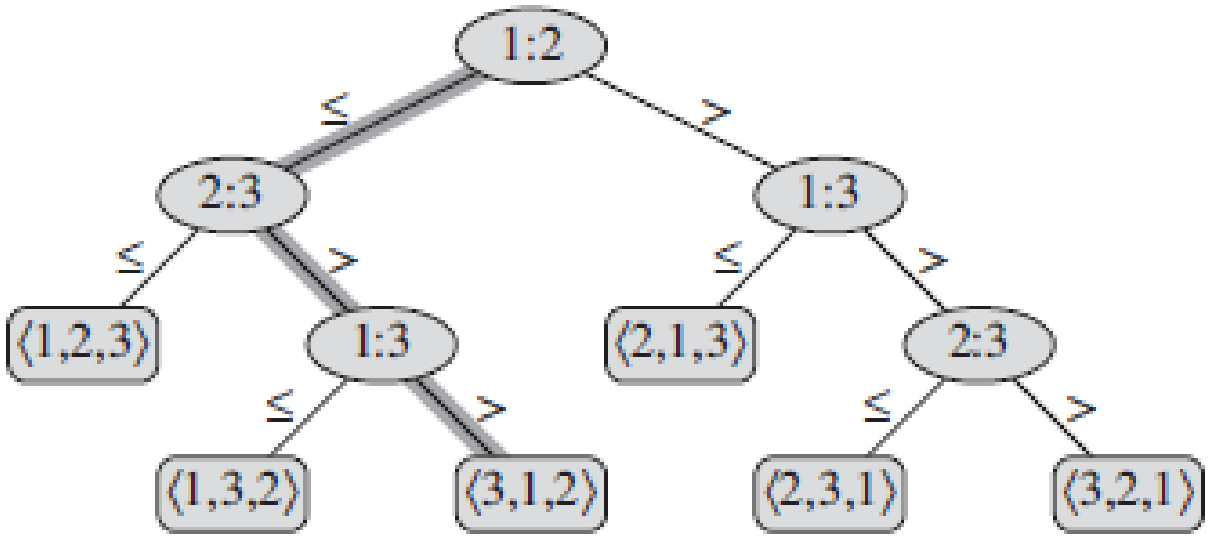
Θεώρημα

Το ακόλουθο δένδρο παρουσιάζει τις συγκρίσεις του insertion sort για ένα πίνακα 3 θέσεων

Στο δένδρο, ο κάθε κόμβος με συμβολισμό i:j παριστάνει τη σύγκριση των a_i, a_j

Ακολουθήστε τη διαδρομή για τον πίνακα

$\langle a_1 = 6, a_2 = 8, a_3 = 5 \rangle$



Θεώρημα

Συνεπώς, για οποιοδήποτε δένδρο αν έχουμε ύψος h και l φύλλα που αντιστοιχούν στην ταξινόμηση n στοιχείων τότε:

- Οι αναδιατάξεις θα είναι n!
- Το ύψος του δένδρου θα είναι 2^h
- Άρα

$$n! \leq l \leq 2^h$$

- Λογαριθμίζοντας παίρνουμε (χρησιμοποιούμε τις μαθηματικές πράξεις που παρουσιάστηκαν στις πρώτες διαλέξεις)

$$\begin{aligned} h &\geq \lg(n!) \\ &= \Omega(n \lg n) \end{aligned}$$

Decrease and Conquer (συνέχεια)

Combinatorial Objects

Οι πιο σημαντικοί αλγόριθμοι αφορούν:

- Αναδιατάξεις (permutations)
- Συνδυασμοί (combinations)
- Εύρεση υποσυνόλων ενός δοσμένου συνόλου

Μαθηματικοί τύποι να υποδεικνύουν το πόσα αντικείμενα πρέπει να παραχθούν

Η αύξηση του πλήθους των αντικειμένων αυξάνει εκθετικά σε σχέση με το μέγεθος του προβλήματος

Το ενδιαφέρον μας είναι στους αλγορίθμους που παράγουν τα αντικείμενα αυτά

Αναδιατάξεις (1/7)

Θεωρούμε πως έχουμε στη διάθεσή μας τους αριθμούς από το 1 μέχρι το n

Θέλουμε να παράξουμε όλες τις πιθανές αναδιατάξεις των αριθμών αυτών

Το πλήθος των αναδιατάξεων είναι n!

Η τεχνική της μείωσης κατά 1 μας υποδεικνύει να παράξουμε τις (n-1)! αναδιατάξεις

Αν επιλύσουμε το μικρότερο πρόβλημα, τότε η λύση είναι προφανής: τοποθετούμε το n σε κάθε μια από τις θέσεις των αναδιατάξεων

Αναδιατάξεις (2/7)

Ο συνολικός αριθμός θα είναι n(n-1)!

Μπορούμε να εισάγουμε το n είτε ξεκινώντας από αριστερά προς τα δεξιά ή από τα δεξιά προς τα αριστερά

Είναι αποδοτικό αν ξεκινήσουμε να εισάγουμε το n στο 1,2,…,(n-1) ξεκινώντας από δεξιά προς τα αριστερά και να αλλάζουμε κατεύθυνση κάθε φορά που μια αναδιάταξη παράγεται

Παράδειγμα

start	1		
insert 2 into 1 right to left	12	21	
insert 3 into 12 right to left	123	132	312
insert 3 into 21 left to right	321	231	213

Αναδιατάξεις (3/7)

Το πλεονέκτημα έχει να κάνει λόγω του γεγονότος ότι αυτή η μέθοδος ικανοποιεί την απαίτηση της μικρότερης αλλαγής (***minimal change requirement)***

- Κάθε αναδιάταξη μπορεί να εξαχθεί από την αμέσως προηγούμενη ανταλλάσσοντας μόνο δύο στοιχεία σε αυτή

Η ιδιότητα αυτή είναι ευεργετική για την ταχύτητα του αλγορίθμου και για τις εφαρμογές που υιοθετούν αναδιατάξεις

Αναδιατάξεις (4/7)

Μπορούμε να αποδώσουμε μια κατεύθυνση σε κάθε αριθμό

Βάζουμε ένα βέλος πάνω από τον κάθε αριθμό

Παράδειγμα: 

Ένα στοιχείο k θεωρείται ότι είναι κινητό αν το βέλος του δείχνει σε ένα μικρότερο στοιχείο (διπλανό στοιχείο)

Στο παράδειγμα το 3 και το 4 είναι κινητά στοιχεία

Με την έννοια του κινητού στοιχείου έχουμε τον αλγόριθμο Johnson – Trotter για την παραγωγή αναδιατάξεων

Αναδιατάξεις (5/7)

ALGORITHM *JohnsonTrotter*(n)

//Implements Johnson-Trotter algorithm for generating permutations
//Input: A positive integer n
//Output: A list of all permutations of $\{1, \ldots, n\}$
initialize the first permutation with $\overleftarrow{1}\ \overleftarrow{2} \ldots \overleftarrow{n}$
while the last permutation has a mobile element **do**
    find its largest mobile element k
    swap k with the adjacent element k's arrow points to
    reverse the direction of all the elements that are larger than k
    add the new permutation to the list

Αναδιατάξεις (6/7)

Παράδειγμα εκτέλεσης του αλγορίθμου (n=3)

Πρόκειται για ένα αποδοτικό αλγόριθμο

Εκτελείται σε Θ(n!) που φυσικά δεν πρόκειται για ικανοποιητική πολυπλοκότητα

Το πρόβλημα δεν είναι ο αλγόριθμος αλλά το ίδιο το πρόβλημα που ζητά την παραγωγή πολλών στοιχείων / αναδιατάξεων

Αναδιατάξεις (7/7)

Ερώτημα

- Μπορούμε να δημιουργήσουμε αναδιατάξεις σε λεξικογραφική / αύξουσα σειρά?

ALGORITHM *LexicographicPermute*(n)

//Generates permutations in lexicographic order
//Input: A positive integer n
//Output: A list of all permutations of $\{1, \ldots, n\}$ in lexicographic order
initialize the first permutation with $12 \ldots n$
while last permutation has two consecutive elements in increasing order **do**
  let i be its largest index such that $a_i < a_{i+1}$ // $a_{i+1} > a_{i+2} > \cdots > a_n$
  find the largest index j such that $a_i < a_j$ // $j \geq i+1$ since $a_i < a_{i+1}$
  swap a_i with a_j // $a_{i+1}a_{i+2} \ldots a_n$ will remain in decreasing order
  reverse the order of the elements from a_{i+1} to a_n inclusive
  add the new permutation to the list

Εύρεση Υποσυνόλων (1/6)

Το knapsack problem είναι ένα παράδειγμα

Για ένα σύνολο στοιχείων $A=\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ θα παράξουμε τα 2^n υποσύνολα

Η τεχνική της μείωσης κατά ένα λειτουργεί ως εξής:

- Το A μπορεί να χωριστεί σε δύο ομάδες: σε αυτό που περιέχει το α_n και σε αυτό που δεν το περιέχει
- Η πρώτη ομάδα είναι το σύνολο των υποσυνόλων των n-1 στοιχείων
- Η δεύτερη παράγεται αν προσθέσουμε το α_n

Εύρεση Υποσυνόλων (2/6)

Παράδειγμα

n	subsets							
0	$\varnothing$							
1	$\varnothing$	$\{a_1\}$						
2	$\varnothing$	$\{a_1\}$	$\{a_2\}$	$\{a_1, a_2\}$				
3	$\varnothing$	$\{a_1\}$	$\{a_2\}$	$\{a_1, a_2\}$	$\{a_3\}$	$\{a_1, a_3\}$	$\{a_2, a_3\}$	$\{a_1, a_2, a_3\}$

Εύρεση Υποσυνόλων (3/6)

Μπορούμε να χρησιμοποιήσουμε μια συμβολοσειρά δυαδικών ψηφίων ως εξής:

- Για τη συμβολοσειρά $b_1, b_2, \ldots, b_n$ το $b_i = 1$ αν το α_i ανήκει στο υποσύνολο και $b_i = 0$ αν το α_i δεν ανήκει στο υποσύνολο

Έχουμε 2^n συμβολοσειρές

Το κενό υποσύνολο συμβολίζεται με το 000 για n=3

Δημιουργούμε όλες τις συμβολοσειρές αν δημιουργήσουμε όλους τους δυαδικούς αριθμούς από το 0 μέχρι το $2^n - 1$

Εύρεση Υποσυνόλων (4/6)

Παράδειγμα

bit strings	000	001	010	011	100	101	110	111
subsets	$\varnothing$	$\{a_3\}$	$\{a_2\}$	$\{a_2, a_3\}$	$\{a_1\}$	$\{a_1, a_3\}$	$\{a_1, a_2\}$	$\{a_1, a_2, a_3\}$

Παρά το γεγονός ότι οι συμβολοσειρές παράγονται σε λεξικογραφική σειρά, τα υποσύνολα δεν μοιάζουν να έχουν ‘φυσιολογική’ σειρά

Εύρεση Υποσυνόλων (5/6)

Η squashed order υποδεικνύει ότι το κάθε υποσύνολο που εμπλέκει το αj θα προκύψει μόνο μετά από τα υποσύνολα που εμπλέκουν τα στοιχεία $\alpha_1, \alpha_2, \ldots, \alpha_{j-1}$

Ερώτημα

- Μπορούμε να παράξουμε τον αλγόριθμο της ελάχιστης αλλαγής ώστε κάθε string να διαφέρει από το προηγούμενο κατά μόνο ένα bit?
- Παράδειγμα:

000 001 011 010 110 111 101 100.

Εύρεση Υποσυνόλων (6/6)

Μια ακολουθία που ικανοποιεί την ελάχιστη αλλαγή στην παραγωγή υποσυνόλων ονομάζεται ***binary reflected Gray code***

ALGORITHM $BRGC(n)$

//Generates recursively the binary reflected Gray code of order n
//Input: A positive integer n
//Output: A list of all bit strings of length n composing the Gray code
if $n = 1$ make list L containing bit strings 0 and 1 in this order
else generate list $L1$ of bit strings of size $n - 1$ by calling $BRGC(n - 1)$
    copy list $L1$ to list $L2$ in reversed order
    add 0 in front of each bit string in list $L1$
    add 1 in front of each bit string in list $L2$
    append $L2$ to $L1$ to get list L
return L